Non-invasive I2C Hardware Trojan Attack Vector

Mohamed Amine Khelif[1,2], Jordane Lorandel[1], Olivier Romain[1]
[1]*ETIS Laboratory UMR 8051, CY Cergy Paris University, ENSEA, CNRS, F-95000 Cergy, France*
[2]*ESIEE-IT, 8 rue Pierre de Coubertin, 95300 Pontoise, France*
{mohamed-amine.khelif, jordane.lorandel, olivier.romain}@ensea.fr

***Abstract*—In smartphones, and more generally in IoT devices, manufacturers focus their efforts on securing communications with the outside world that are more exposed to attack while considering communications between secure components. By doing this, it results in internal communication buses with little or no security against attackers. I2C is the most used internal communication bus in IoT devices to communicate with sensors and memories. It is also used in recent smartphones to connect the Trusted Execution Environments (ARM TrustZone, Apple SEP, or Google Titan M) to a dedicated EEPROM memory that contains secret information such as encryption keys, anti-replay counter, or the boot ROM.**
In this paper, we propose a non-invasive attack through a hardware trojan on the I2C bus, which will allow us to perform two attack scenarios: a heart bleeding type attack which will allow retrieving additional information at each memory read, and a buffer overflow attack which will allow writing additional data in the memory at each write which can result in modifying secret information such as password or counters. These attacks can be performed on any device using the I2C bus. In the context of smartphones, these attacks will allow the extraction of sensitive information stored in the secure EEPROM memory.

***Keywords*-Hardware attack, Hardware Trojan, Security, I2C, smartphones, IoT.**

I. Introduction

Smartphones have been massively adopted by consumers for the last decades. However, there are still many challenges related to security, especially for these devices containing a lot of sensitive and personal data. In 2020, the number of smartphone users worldwide exceeds 3.6 billion [1], representing more than 30% of all circulating non-IoT devices [2]. On the other side, the number of IoT devices also grows these last years, up to 11.7 billion in 2020 [2]. This proliferation and the amount of personal information contained in these devices make them a privileged attack target for hackers.
In order to protect these devices, a new standard was created under the name of Trusted Execution Environments (TEE). They work under a simple idea of isolating trusted security applications from other applications into two execution environments. Figure 1 represents TEE's simplified architecture.

As we can see in this figure, the Rich Execution Environment which is abbreviated as REE in literature is the Normal World. This environment has its operating system (OS), iOS, or Android for the most popular ones, and executes client applications. On the other side, we have the TEE which is isolated from the REE in three different forms depending on the vendor: virtual processor, internal co-processor (inside the SoC), or external co-processor. The TEE executes only trusted applications and it has its own OS and dedicated components (depending on the vendor) such as crypto-engine, security sensors, dedicated memory, etc. . It also shares components with the REE such as various memories, power management unit (PMU), Clocks, Crypto-engine, etc. . All the exchanges between the TEE and these components are performed using communication protocols. The most used are I2C and SPI protocols. I2C bus, in particular, is used as a data bus between the dedicated EEPROM memory and the TEE for example in iPhones [3].

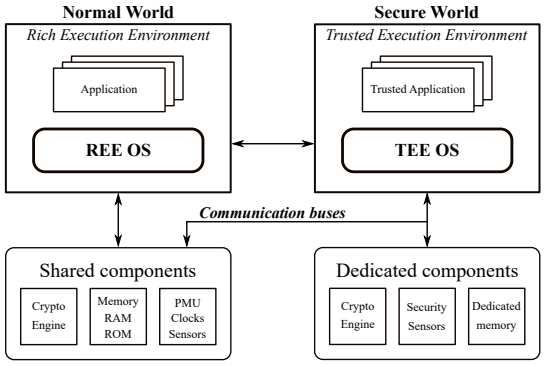

Fig. 1. Simplified architecture of a TEE principle.

In this paper, we study the impact of an I2C hardware trojan attack using a new semi-invasive approach based on clock and data signal control. The attacker will be connected to the I2C bus and will act as an invisible malicious component that modifies the communication in real-time. Two attacks will be thus performed: a heartbleed attack by reading more information from the memory than requested by the master, and a buffer overflow by writing additional data into the memory. These attacks are performed on a test platform composed of an STM32 micro-controller communication with EEPROM memory. The choice of this set-up was motivated by two main constraints: being as close as possible to a TEE/EEPROM communication behavior and control the transaction in order to verify the results of the attack. In addition, STM32 is one of the most used micro-controllers in IoT devices using the HAL standard I2C communication library, which makes the impact of these attacks potentially important.
This paper is organized as follows. First, section II presents

978-1-6654-1609-2/21/$31.00 ©2021 IEEE

the different technologies of TEE used in smartphones and IoT devices with a focus on Apple's Secure Enclave security mechanisms and state-of-the-art attacks on communication buses. Then, section III details the I2C protocol and the implementation of the different attack scenarios. Finally, experimental results are given in section IV before the conclusion in section V.

II. Related Works

A. Smartphones security

In smartphones, security is managed by a Trusted Execution Environment. TEEs are secure execution environments that monitor the integrity and security of the system and user's data [4]. Among the various tasks that are assigned to them, they mainly have to ensure a secure boot of the device through a bootRom, ensure the confidentiality of personal data by performing encryption and decryption, as well as the protection of the different counters and passwords of the smartphone.

There are three types of TEE in the market based on different technologies [5]:

Virtual Processor: the most widely known is the ARM TrustZone (TZ) [6], which operates by adding an additional privilege level in addition to the classic levels (user, kernel, and hypervisor). This new level is called Secure Monitor and has two execution states: secure and non-secure. The ARM TrustZone and the main system share the different controllers and devices of the SoC. The TZ also have a dedicated operating system called TEE-OS, a Unique ID (UID) of the smartphone fused in the SoC, and a dedicated I2C and SPI bus that are used to connect the TZ to a dedicated EEPROM, to a fingerprint sensor, etc. . These shared components depend on the smartphone architecture.

Internal Co-Processor: this technology is used by Apple in their smartphones with the name of Secure Enclave Processor (SEP) [7]. In iPhones [3], the SEP manages security, protection keys (password, fingerprint, faceID, GID, UID...) and data encryption/decryption. It uses encrypted memory and integrates a hardware true random number generator (TRNG). It has its own operating system, SEPOS, as well as its own devices (crypto-engine and random number generator) and its own hardware inputs and outputs (GPIO, UART, etc.). It also has shared hardware as a PMU, a memory controller, a clock generator, and the RAM. It is also connected to a dedicated EEPROM memory through the I2C bus. Communications between the Secure Enclave and the application processor are limited to an interrupt triggered mailbox.

External Co-Processor: such as Google Titan M processor which is used from 2018 in pixel 3. Titan M [8] is used as an interposer between the SoC and the boot flash in order to verify the boot firmware loaded. To do that, the processor has only dedicated components such as a cryptography co-processor, a hardware TRNG, a RAM, a ROM, a flash memory, and several communication buses such as I2C and SPI. It also embedded a dedicated operating system and a memory built-in self-test to guarantee the integrity of the processor.

B. Apple iPhone's SEP as a case of study

Apple iPhones are known to be the most secure smartphones on the market thanks to its TEE security system Secure Enclave. As said previously, the secure enclave processor is an internal co-processor integrated into Apple's devices from Apple A7 SoC. A lot of information about about the operation process of the SEP can be retrieved from Apple patent [9], iOS security guide [7] [10] and other research and reverse engineering studies [3] [11]. Figure 2 gives a simplified overview of the interaction of the SEP with the application processor (AP) and the other peripherals.

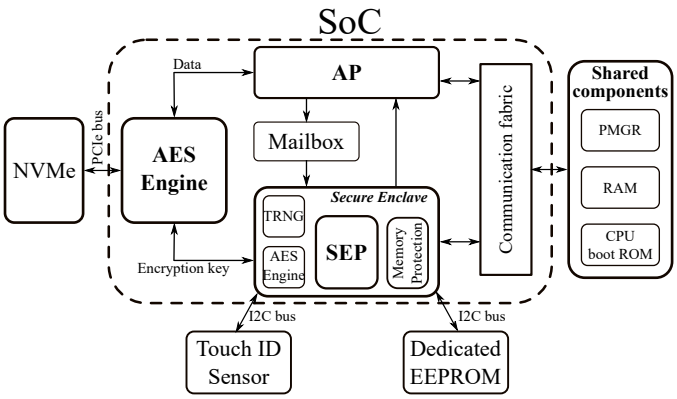

Fig. 2. Simplified architecture of Apple's SoC and interaction between AP and SEP.

The Secure Enclave (SE) is physically isolated from the AP inside the SoC and communicates through a mailbox. The SE is based on the Secure Enclave Processor which is a 32-bit ARMv7 processor working under a specific operating system called SEPOS [3]. Secure Enclave also has an integrated secure AES engine and hardware TRNG used to guarantee the security of the SEP and generate encryption keys by deriving them from various keys such as UID, GID (both are fused in the SoC), and user password. It also integrates various hardware protocols such as JTAG, UART, SPI, I2C. For JTAG and UART they are deactivated in production smartphones in order to mitigate the security risks. For SPI and I2C, they are used to communicate with dedicated components outside of the SoC such as a dedicated EEPROM or various security sensors like touch ID and FaceID. This EEPROM is also called anti-rollback memory and it ensures the protection of the password attempt counter and the secure boot. The Secure enclave shares multiple components with the application processor such as boot ROM, PMGR, and an encrypted RAM partition in order to prevent attacks from malicious applications. Finally, the SE protects the data stored in the non-volatile memory (NVMe) by providing encryption keys that will be used to encrypt the data before writing them in the memory inside. This mechanism ensures the protection of personal information against off-board memory reading and bus listening besides high communication rates of the PCIe bus.

C. Attacks on communication Bus

In Trust Zones, and more generally in SoCs used in IoT devices and smartphones, the utilization of data buses to communicates with other chips of the device is of utmost importance. These buses rely on commonly used protocols that already have been the target of attacks more or less developed. From the most used we have:

1) I2C: Inter-Integrated Circuit protocol is used in smartphones and in IoT devices in general, as a communication bus between the processor/microcontroller and various sensors and/or EEPROM memory. The main attack performed on this bus targets the clock signal which has a critical role in a synchronous protocol such as I2C. These clock glitching attacks [12] affect the system and result in a global malfunction or a denial of service. I2C can also be targeted by attacks that aim to recover the data and firmware stored in an EEPROM [13] [14].

2) SPI: Serial Peripheral Interface is also one of the most used communication buses in smartphones and IoT devices. It is also very similar in peripheral type usage as I2C, except that SPI is faster and requires more space for implementation (one additional wire per slave). The bus is also vulnerable to attacks that aim to recover data and firmware from components [13].

3) PCIe: Peripheral Component Interconnect express is a high-performance communication bus originally used in computers to connect high-speed components such as GPU, WiFi cards, or SSD to the motherboard. Nowadays, it is used in iPhones from the 6s version and newer as an internal bus connecting the SoC to the non-volatile memory. As demonstrated by [15] and [16], this bus can also be targeted by hardware attack in order to passively extract information using a sniffer approach or actively interacting with the packets exchanged using a Man-in-the-Middle.

4) UART: Universal Asynchronous Receiver-Transmitter which can be an external component or integrated into the chip which allows two components to communicate without using a clock. This bus has many functionalities depending on the device and that can be exploited by an attacker, from which we denote debugging, get an unauthenticated root shell, or even access the boot loader [13]. UART is generally poorly protected which makes the attack not very difficult to undertake.

5) JTAG: is a protocol developed by the Joint Test Action Group and which is named after. The JTAG is used for testing, programming, debugging, and validation of devices [17]. This protocol is embedded from the most basic IoT device to the most evolved smartphone. As with previous protocols, JTAG is also targeted by attacks such as fault injection [18] in order to gain privilege escalation. Considering its usage and the attack opportunities allowed by this protocol, the manufacturer deactivates it at the production step in order to limit the risks. For example, in iPhones, this bus is disabled, but recently it can be enabled by exploiting a checkM8 vulnerability [19].

6) USB: Universal Serial Bus is used as the wired communication protocol used in smartphones for different purposes such as interfacing/communicating with a computer and/or recharge the device. This bus has also been targeted by various attacks, the most known are Juice jacking attacks. These types of attacks consist of integrating a malicious device in a USB charging connector that will inject malware in a smartphone [20].

Being the most used communication protocol in IoT and non-IoT devices, the I2C bus represents a valuable source of information for attackers. In recent smartphones, it is used to connect the TEE to the secure EEPROM memory. This memory store critical information such as password counter, encryption keys, etc.

In the state of the art, researchers focus on clock glitching the I2C clock, sniffing the communication, and randomly modify the payload in order to generate errors [14]. With a hardware trojan attack based on controlling both SCL and SDA lines that we propose, we can perform more advanced scenarios comparing to the state of the art. with our approach, we can read or write the memory or any other device without modifying the hardware to avoid detection, simply by probing the lines through pull-up resistors.

III. Methodology

A. I2C protocol

The need for a universal bus that would be simple, inexpensive, space-saving, and easy to implement led Phillips to create the I2C bus in 1982 [21]. This bus allows connecting several peripherals together with relatively low communication rates. The classic architecture consists of a single master with multiple slaves that can be sensors or memories. It is also possible to have multi-master configurations with arbitration to control the communication. Initially, the I2C specification defined a maximum clock frequency of 100 kHz, later increased through three modes: fast with 400 kHz, high speed with 3.4 MHz, and ultra-fast 5 MHz. Although it is less efficient than the SPI bus in terms of throughput, the I2C has the main advantage of having a small surface requirement thanks to the fact that it is composed of only 4 wires (VCC, GND, SDA, and SCL) for a maximum of 128 slaves (in the 7-bit address configuration). This feature spreads its utilization in many IoT and non-IoT devices.

Two types of communication are possible in I2C, allowing to write data into the slave and to read data from the slave. For read communication, there are three main types of packets:

- Current Address Read: read-only one byte of payload from the memory address accessed by the previous packet.
- Sequential Read: read n bytes of payload from the memory address accessed by the previous packet.
- Random Read: this sequence is composed of two packets: a first write packet without payload in order to update the last memory address accessed then followed by a read packet with n byte payload.

For write communication, there are two types of packets:

- Byte Write: write only one byte of payload size.

- Page Write: write n bytes of payload, with n is the size of a memory page in the case of an EEPROM.

Figure 3 represents the different communication format in I2C. Write packets includes Byte and Page write, and read packet includes current address and sequential read. For random read as it is a sequence composed of two packets: write packets without payload and a read packet separated by a restart signal, it is represented separately in the figure below.

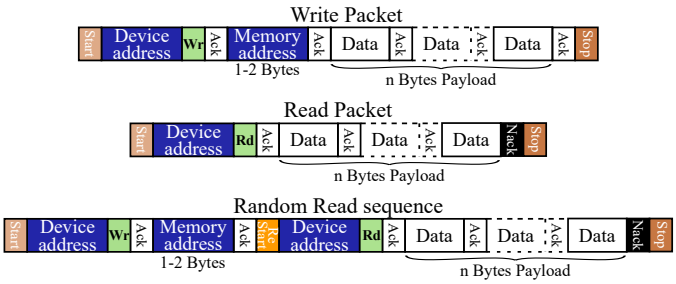

Fig. 3. I2C communication frame types.

The communication on the I2C bus can only happen between the master and a slave device, at the initiative of the master. It imposes a clock signal on the SCL wire by driving the line to 0 and releasing it to 1 in order to generate the signal. When the clock is generated the communication can start following these steps:

Idle: the bus is not used, SDA and SCL lines are in a high state.

Start condition (and restart or repeated start): when the master wants to communicate with the slave after the idle state, it generates a start condition by first driving the SDA line low, while SCL remains high. For a restart condition, it is similar to a start condition in driving signals, the master directly sends another packet without stopping the communication to avoid conflict with potential other masters.

Device address: the master sends 7 bits corresponding to the device slave address.

Read/Write bit: the slave address is directly followed by the R/W bit that will define the rest of the packet format.

Ack/Nack: each byte is followed by an Ack/Nack bit. For write packets and if there is no error in the communication, the slave will acknowledge each byte. For read packets, the slave acknowledges the first byte (device address and R/W bit) then the master will acknowledge every byte received from the slave until the last byte where the master sends a negative acknowledgment bit to stop the slave from sending data.

Memory address: this information is only available in write packets, and depending on the type of the peripheral it can be one byte (ex. temperature sensor) or two bytes long (ex. EEPROM).

Payload: is the data in the packet.

Stop condition: when the communication is over, the master generates a stop condition by releasing first the SCL line and then the SDA line.

B. Experimental set-up

The experimental set-up is composed of an STM32 Nucleo-F446RE kit communicating with an AT24C128 EEPROM 128kB memory from ATMEL through the I2C bus. In order to know precisely the data contained in the memory, the STM32 initializes each page with different data using a sequence of the page write, with a 64 bytes payload size (corresponding to one page in this EEPROM). Then, the micro-controller starts the communication with the memory by using a sequence of reads and writes that we target through our attack. For verification, we use also a Digilent Analog Discovery 2 to monitor signal with both the scope and the protocol decoder features. The experiment set-up is represented in Figure 4

The architecture of the attacker was implemented onto an FPGA Zedboard platform from Digilent, and the design runs at a clock frequency of 100MHz in the PL part.

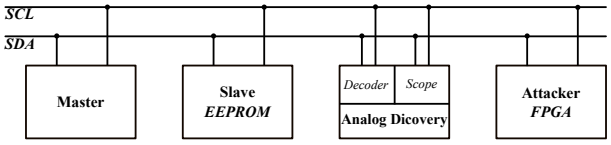

Fig. 4. experimental set-up architecture.

C. Approach

The proposed approach is based on two phases: a sniffing phase and an attack phase which can be one of the two proposed scenarios.

The sniffing step is performed by the FPGA in order to retrieve the necessary information about the system such as:

Master/Slaves addresses: this information can be found by sniffing the first 7 bits of each packet new packet and it will be necessary to know all slave addresses on the bus.

Number of slaves: from the previous information it will be easy to determine the number of slaves because each one has a unique address on the bus

Memory addressing format: to identify a random read sequence involving the targeted slave. To do that, we need to find in the logged communication a write packet followed by a read packet, separated by a restart condition. The size of the write packet will then provide information about the memory addressing format. If the packet is two bytes long the memory address format is one byte. If the packet is three bytes long then the memory address is on two bytes. Two bytes address format corresponds to a memory address which means that the slave is a memory, in our case an EEPROM.

Sniffing the I2C communication can be performed by many devices such as the protocol decoder of the analog discovery 2, but we choose to develop an FPGA architecture in order to synchronize attack scenarios with particular events on the bus. This can also be used by attackers to perform an offline depth analysis of the recorded packets with advanced algorithms in order to find sensitive information even in encrypted communications.

In our case we decided to perform the following attacks:

1) Heartbleed attack: This attack will be used to over-read the memory by controlling SDA and SCL. In order to not create conflict on the bus and been stopped by an unfortunate stop condition generated by the master, the attack will be performed by corrupting a legitimate read packet from the Master. The principle of this attack is to perform an on-the-fly analysis of the first byte of each packet looking for a read packet sent to the targeted slave. Then, from the acknowledgment of the first byte (device address + R/W bit), the FPGA attacker will control both SCL and SDA lines. For SCL, in order to avoid the release of the clock when the master has finished reading data, the FPGA attacker will generate a synchronized clock with the original one. For the SDA line, ack/nack bit sent by the master bit will be set to zero, forcing the communication to continue. Then, the memory will answer all the stored information byte by byte and page by page.

2) Buffer overflow attack: This attack will allow overwriting a section of the memory in order to replace security information in the memory such as encryption keys, counters, etc. The principle of the attack needs as for the previous one, to analyze the communication and to take control of both the SDA and SCL lines. We first analyze the communication looking for a write packet, and then from the acknowledgment bit of the first byte, we generate our synchronized clock on SCL and force SDA to zero. This will avoid the generation of stop condition and making possible to write data to the slave up to one page, which in the case of our memory is 64 bytes.

In a practical approach, this I2C hardware trojan can be inserted into an IoT device without modifying the device. In the case of these devices, the attacker will need to locate I2C pull-up resistors and interface directly by soldering wires on them or using a probing machine in order to be non-invasive.

IV. Results and discussion

To demonstrate the effectiveness of the proposed attacks, we realized the following experiments:

A. Sniffing I2C communication

For this experimentation, we decided to add another slave (temperature sensor) in the presented set-up in order to detect and define the specification of each device. Table I resume the information extracted from 30 s of I2C sniffing. The first information that we got from sniffing is the presence of two slaves because of the two different device addresses 0x18 and 0x50 with respectively 8 bits and 16 bits memory addressing format. This information is determined from the memory address of the write packet in a random read sequence. Memory addressing format gives also information about the type of slave. For 8 bits format, the slave may be a sensor because of the small number of registers embedded, and for 16 bits format, it probably corresponds to a memory chip because of the need for more addressing space. These assumptions can also be correlated to the known I2C slave address database which lists the known devices using their address [22]. For addresses, 0x18 only accelerometers and temperature sensors are listed, and for 0x50 it can be a memory or a PWM driver (not so common in IoT devices).

TABLE I
SPECIFIC METRICS EXTRACTED FROM I2C RECORDED TRAFFIC USING THE SNIFFER.

Information from analysis	Slave 1	Slave 2
Address	0x18	0x50
Memory format	8 bits	16 bits
Byte Write	No	Yes
Page Write	Yes	Yes
Current Address Read	No	No
Sequential Read	No	No
Random Read	Yes	Yes

We analyze also the type of packet used for each device in order to ensure that the attacks can be performed: read packets for heart bleeding and write packets for buffer overflow. This architecture can also be used to sniff all the packets in order to perform a communication analysis and data mining relayed on multiple parameters such as the number of packets, the time between packets, payload size, etc.

B. Heart bleeding

Heart bleeding attack aims to extract more information from memory than asked by the master by a read request. To achieve this goal, the attacker must control SDA and SCL. Figure 5 shows the experimental results of the attack recorded by the analog discovery 2 protocol analyzer connected to the I2C bus. The first graph represents the normal read request sequence between the master and the memory, with a write packet to update the memory address and a read packet with 4 bytes of payload. The second graph represents the result of the attack, which forces all ack/nack bit to zero in order to avoid the negative acknowledgment that stops the memory. The attacker regenerates the clock by driving SCL to zero. This attack permitted us to extract all the data stored in memory.

When performing the preliminary tests of the attack, we noticed that the I2C clock generated by the master is at 100 kHz but the duty cycle is not at 50%. It is 47% in the high state and 53% in the low state, which has to be taken into account in order to generate a correct clock for the attack. In fact, to avoid some glitches occurring when regenerating the clock with the FPGA, we chose to generate a clock that will have a ratio of 43% in the high state and 57% in the low state. This modification will make the falling edge occurring sooner than normal in order to have a more stable SDA signal and reduce the chances to be stopped by a potential stop condition sent by the master. It could be also interesting to explore the degree of resilience of the system when the high state is reduced as low as possible, and what impact will it have on the system.

C. Buffer overflow

Figure 6 shows the experimental results of the buffer overflow attack. This attack will overwrite a complete page of the memory even if the attacker wants to write only one byte. the attack is limited to one page (64 bytes) because of the limitation of the memory that needs to temporize between

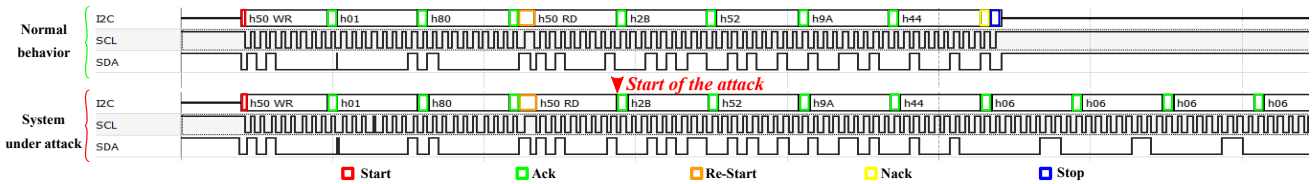

Fig. 5. Experimental results of heart bleeding attack (recorded with analog discovery 2 Logic analyzer).

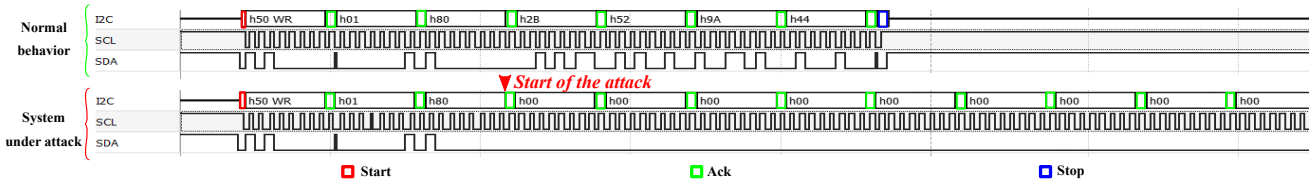

Fig. 6. Experimental results of buffer overflow attack (recorded with analog discovery 2 Logic analyzer).

writing two pages. In Figure 6, the first graph represents a normal 4 bytes payload write packet, and the second graph is the result of the attack. In order to perform the attack, the FPGA forces the SDA line to zero and generates the SCL clock from the first ack bit after the memory address. This control of SDA will corrupt the data stored in the memory and makes it impossible a stop condition. This attack will overwrite up to one page of the memory, depending on where the master starts writing.

V. Conclusion

In this paper, we proposed a hardware trojan capable of sniffing and attacking the I2C bus. This bus even massively used in IoT and non-IoT devices are still poorly secured. It is also used as a communication bus between the TEE and its secure memory in recent security mechanisms of smartphones. The architecture proposed proves that an attacker device connected to the I2C bus can interact with the system by manipulating data and clock signals in order to extract all the data stored in the memory through a heart bleeding attack or to overwrite up to one page of the memory through a buffer overflow attack. These attacks could be performed without being detected by neither the master nor the slaves. If the memory is not encrypted, the attacker can directly access the secrets otherwise the sniffed communication can be analyzed in order to extract other relevant information.

In future work, the architecture will be improved to perform a hardware man-in-the-middle attack, in order to fully control the bus in order to analyze and modify data on the fly. We will also study the countermeasures against our attack, such as analyzing bus behavior by the master or through implementing a dedicated chip capable of recognizing attacks.

References

[1] S. O'Dea, "Number of smartphone users worldwide from 2016 to 2026," Online, https://bit.ly/3fp82Ag, Mars 2021, accessed: 2021-05-25.

[2] L. S. Vailshery, "Internet of things (iot) and non-iot active device connections worldwide from 2010 to 2025," Online, https://bit.ly/2SwKVuB, Mars 2021, accessed: 2021-05-25.

[3] T. Mandt, M. Solnik, and D. Wang, "Demystifying the secure enclave processor," *Black Hat Las Vegas*, 2016.

[4] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[5] M. Peterlin, A. Adamski, and J. Guilbon, "Breaking samsung's arm trustzone," *Blackhat US*, 2019.

[6] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[7] M. Gulati, M. J. Smith, and S.-Y. Yu, "Security enclave processor for a system on a chip," September 2014, uS Patent 8,832,465.

[8] R. Triggs, "Will google's titan m make it harder for the roming scene?" Online, https://www.androidauthority.com/titan-m-security-chip-915888/, October 2018, accessed: 2021-05-25.

[9] M. Gulati, M. J. Smith, and S.-Y. Yu, "Security enclave processor for a system on a chip," U.S. Patent US8 832 465B2, September 09, 2014. [Online]. Available: https://patents.google.com/patent/US8832465B2/en

[10] A. P. Security, "Secure enclave," Online, https://support.apple.com/fr-fr/guide/security/sec59b0b31ff/web, May 2021, accessed: 2021-05-25.

[11] M. Renard, "Investigation numérique & terminaux apple ios: Acquisition de données," in *SSTIC conference*, 2014.

[12] F. Gomez-Bravo, R. J. Naharro, J. M. García, J. G. Galán, and M. Raya, "Hardware attacks on mobile robots: I2c clock attacking," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 147–159.

[13] A. Gupta, *The IoT Hacker's Handbook: A Practical Guide to Hacking the Internet of Things*. Apress, 2019.

[14] A. Jha, "Iot security - part 16 (101 - hardware attack surface: I2c)," Online, https://bit.ly/3pGsH6C, September 2020, accessed: 2021-05-25.

[15] M. A. Khelif, J. Lorandel, O. Romain, M. Regnery, D. Baheux, and G. Barbu, "Toward a hardware man-in-the-middle attack on pcie bus," *Microprocessors and Microsystems*, vol. 77, p. 103198, 2020.

[16] M. A. Khelif, J. Lorandel, O. Romain, M. Regnery, and D. Baheux, "A versatile emulator of mitm for the identification of vulnerabilities of iot devices, a case of study: smartphones," in *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, 2019, pp. 1–6.

[17] A. Sguigna, "Mitigating jtag as an attack surface," in *2019 IEEE AUTOTESTCON*. IEEE, 2019, pp. 1–7.

[18] F. Majéric, B. Gonzalvo, and L. Bossuet, "Jtag fault injection attack," *IEEE Embedded Systems Letters*, vol. 10, no. 3, pp. 65–68, 2017.

[19] axi0mX, "Checkm8: Open-source jailbreaking tool for many ios devices," Online, https://github.com/axi0mX/ipwndfu, November 2019, accessed: 2021-05-25.

[20] B. Lau, Y. Jang, C. Song, T. Wang, P. H. Chung, and P. Royal, "Mactans: Injecting malware into ios devices via malicious chargers," *Black Hat USA*, 2013.

[21] S. B. Jean-Marc Irazabal, "I2c manual" application note," NXP Semiconductors, Tech. Rep., 2003.

[22] J. Romkey, "I2c address list," Online, https://i2cdevices.org/devices/pca9685, March 2020, accessed: 2021-05-25.